IMPROVED RAY TRACING PERFORMANCE THROUGH TRI-ADAPTIVE SAMPLING

By

Tristan Craddick

A Project Report Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science

University of Alaska Fairbanks

May 2020

APPROVED:

Dr. Orion Lawlor, Committee Chair
Dr. Jon Genetti, Committee Member
Dr. Jonathan Metzgar, Committee Member
Dr. Chris Hartman, Chair
Department of Computer Science
Dr. Bill Schnabel, Dean
College of Engineering and Mines
Dr. Michael Castellini, *Dean of the Graduate School*

Abstract

Ray tracing is a technique capable of rendering high quality images by tracing rays from the camera position into the scene and examining the points they intersect with. With the advent of NVIDIA RTX hardware, improving renderer design through greater algorithmic efficiency will allow for even greater real-time rendering capabilities. Naïve implementations are simple to implement and cheap enough to run well on modern systems, but often have issues with aliased edges due to lower quantities of rays for scene sampling. Techniques such as super-sampling are capable of reducing or entirely eliminating aliasing, but carry a high-performance cost due to additional ray requirements. Under-sampling is a technique that allows a single ray to determine the color of multiple pixels, allowing for high performance in regions of little variation. The combination of these techniques is collectively referred to as Adaptive Sampling. Our implementation of this algorithm operates by rendering the scene at a low resolution and then sampling the resulting image to determine if rays are necessary at higher resolutions. In this project, we implement a form of this multiple-resolution approach based upon a triangular grid overlaying the pixel grid. Results on RTX cards indicate a performance increase of 29-40% over the naive renderer, and a 1-4% increase over the traditional adaptive sampling algorithm, all while achieving little degradation in quality compared to the ground truth image.

Table of Contents

1 Introduction

Modern applications in computer graphics require a high degree of physical accuracy and quality while remaining efficient enough to allow for real-time rendering. The ray tracing rendering technique has often been the target of research, but until recently has proven too costly for use in real-time applications. With the release of NVIDIA's real-time ray tracing (RTX) platform, hardware level ray tracing has improved performance by a significant enough margin to warrant usage. With the increase in usage resulting from this framework, it is valuable to re-examine the algorithms commonly used with ray tracing and seek further efficiency improvements.

Ray tracing as a rendering technique has long been known to provide high quality physically based images. Simple implementations can allow for cheaper performance costs, but have drawbacks such as poor shading and aliased edges. There are several approaches that have been designed to reduce or eliminate issues with aliasing in ray traced scenes. Most, however, create a high performance cost due to the necessity of large quantities of rays per-pixel, as in distributed ray tracing models.

One such field of research is a technique known as adaptive super-sampling. Through this process, pixels can be effectively subdivided as necessary depending upon the rate of change between pixels and subsequent sub-pixels. Rays can then be used to determine the concrete color of the sub-pixels, the results of which can be integrated together to recreate the true color of the full pixel. This relationship can be directed in reverse as well, in a process called under-sampling, allowing a single ray to accommodate the color of multiple pixels in regions of little change. Together, these techniques are collectively referred to as adaptive sampling. One manner of implementing adaptive sampling involves the use of multiple grids of various resolutions ranging from coarser variants of the target resolution to increasingly fine multiples of the target resolution. The application of this multi-resolution adaptive sampling approach allows for improved anti-aliasing in the rendered images, with improved

rendering times over naively sampled and super-sampled renders.

In this project we propose a modification to the multi-resolution approach. Rather than adhere to the rigid structure of the quad-tree grid that is inherent to textures and stored pixels, a triangular grid is utilized to collect samples. This triangular grid is effectively overlayed the traditional texture format, used to collect samples in an offset pattern, and then mapped back to the traditional grid for display to the screen. We believe that usage of this triangularized sampling will require fewer samples, improving algorithm efficiency without quality degradation in the resulting render.

2 Related Research

Ray Tracing as it is commonly known today was proposed as a rendering technique in the 1980's by Turner Whitted, as a recursive approach that allowed for great physical accuracy in scenes despite the required processing time (*Whitted*, 1980). Common sources of aliasing were noted, ranging from sharp changes in intensity to the use of textures. Notably, Whitted discussed the fact that these aliasing effects were caused by under-sampling these volatile regions, and that subdividing these pixels could provide the accuracy necessary to properly interpolate between colors and reduce aliasing effects.

The aliasing issue was not purely resolved at the time of Whitted's research however, and much of the research to follow was directed at improving image quality while reducing the number of rays necessary to achieve such quality. The use of uniformly distributed rays is documented to result in predictable noise in the rendered image, which can be effectively reduced or eliminated through the use of non-uniform distributions of rays such as described by the tenets of stochastic sampling (*Cook et al.*, 1984), (*Cook*, 1986). By coupling improved distributions of rays with the super-sampling practices proposed by Whitted, much of the noise can be processed to the point that it is eliminated when constructing final pixel colors from interpolated values (*Mitchell*, 1991).

These techniques are all categorized as image-space as the decision whether to subdivide each pixel is made based primarily upon the colors returned by each sub ray. As a whole, these techniques are still known to have issues with detecting small objects or very thin objects. There are approaches to deal with this, categorized as object-space, that attempt to examine how a ray hit an object, or pass by an object by detecting intersections with bounding volumes. One approach involves using cone tracing in order to examine all possible objects in a scene that would project onto a pixel, and then determining whether it is necessary to subdivide based upon how marginal the closest intersection with an object is (*Genetti and Gordon*, 1993). This process allows arbitrary small objects to be detected based

upon a subdivision limit, while still retaining high efficiency for regions of inactivity within a scene. The concept of including dimensions and qualities beyond color is expanded on by others, with frameworks existing that use multidimensional functions to allow for effects such as soft shadows, motion blur, and other effects (*Hachisuka et al.*, 2008).

By exploiting the nature of pixel subdivision, the relation between coarse pixel blocks for under-sampling and fine sub-pixels for super-sampling can be represented through grids of varying resolution. This process is utilized by Lawlor and Genetti in an expansion upon the previous object space renderers, allowing for time compressed rendering through a sample-or-interpolate process (*Lawlor and Genetti*, 2014). By checking the color variation within the coarser grid against an error metric, it can be determined whether the use of finer resolution grids is necessary. In effect, each grid can be imagined as a 2D image with specified resolutions for processing, allowing for highly efficient use of the GPU for primary processing and secondary effects.

This concept of sampling the scene at varying resolutions has been explored from several angles such as modifying resolution based on viewports or pipeline passes (*Binks*, 2011), (*McFerron and Lake*, 2018), (*He et al.*, 2014). The idea of resolution grids can be most closely likened to variable rate shading (*van Rhyn*, 2019), which has risen in popularity recently with hardware support by Intel and NVIDIA (*Lake et al.*, 2019a), (*NVIDIA*, 2018a). There are known performance benefits arising from usage of this technique (*Lake et al.*, 2019b), so benefits are expected out of our research into ray tracing with multiple resolution grids. If combined with other adaptive techniques such as temporal aliasing, it is likely that further benefits can be observed with little degradation in quality (*Xiao et al.*, 2018).

The efficiency improvements of adaptive sampling are not to be understated, with some implementations showing framerate boosts of up to four times given specific sample quality thresholds (*Lawlor and Genetti*, 2014). Approaches that make use of both image and object-space adaptive sampling allow for efficient, alias-resistant renders even for scenes with complex geometry and textures (*Bongjun Jin*, 2009).With recent advancements in hardware

allowing for hardware-based ray tracing, effective technique will allow for production-quality images to be rendered in real-time applications. Even the otherwise unexpected target of raytracing on mobile devices is possible through effective under-sampling techniques (*Won-Jong Lee*, 2016), (*Kim et al.*, 2016). This also allows for more complex ray tracing algorithms that deal with indirect lighting to be utilized more effectively (*Křivánek et al.*, 2014), as the increased cost is offset by shooting fewer rays over the frame.

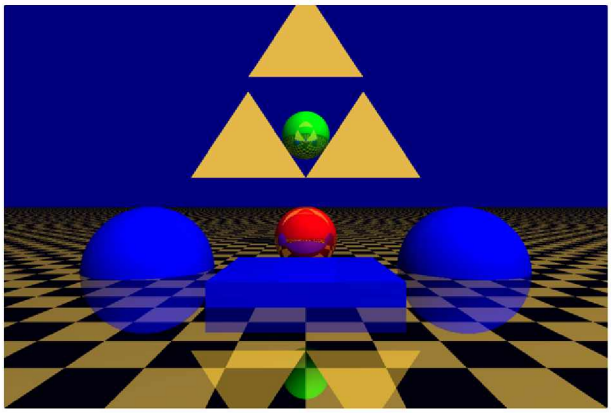


Figure 3.1: Basic Scene Composition

3 Problem Analysis

Adaptive sampling is an algorithm composed of two processes related by the level of detail required in the scene. In regions where there is little detail and relatively little change from pixel to pixel, it would suffice to use one ray to represent larger blocks of pixels. Conversely, in regions where there is a greater rate of change between pixels, more than one ray per pixel may required to accurately determine what the proper pixel color should be.

Equal in importance to the theory of the algorithm is understanding how it works in practice. Take the render of a demonstration scene in Figure 3.1. Though the geometry in the scene is simplistic, there is a fair amount of detail to look through, compounded by the checkerboard pattern of the floor texture.

In this scene, we can highlight all regions where super-sampling is necessary to determine the appropriate color for a pixel. Phrased alternatively, the areas highlighted in green in Figure 3.2 represent pixels where one ray per pixel is not enough to accurately determine the pixel colors. Notably, highlights lie along the edges of objects, or edges of the checkerboard pattern, where aliasing would be expected to occur.

The same image can be used to understand where under-sampling is possible as well. Note that the majority of the object compositions that are not actively reflected upon by other objects in the scene remain an equivalent, or very similar color; likewise for the floor

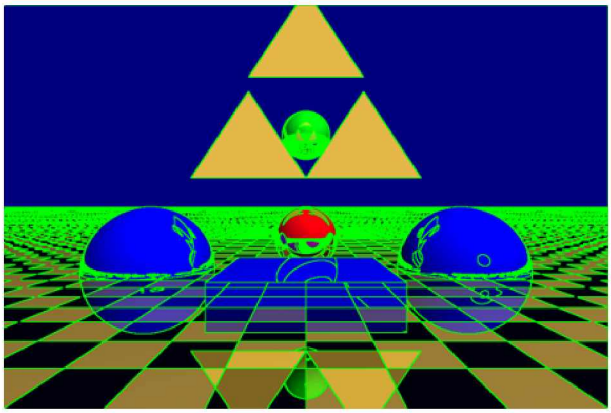


Figure 3.2: Highlights For Super-Sampling

panels and the otherwise blank sky. Taking a look at the sky for instance, it is one solid color throughout the entirety of the image. There are no obstructions, there is no gradient to the color, there is just one solid color throughout. With this in mind, there is no need to spend a great number of rays to determine that, indeed, one pixel of the sky is the same color as the neighboring pixel. Rather, one ray can be used to determine the color of an entire region of pixels in the resulting image. This can be likened to how our eyes look for contrast in our surroundings. If the wall in a room is a solid color, we only need a quick glance to determine that there is nothing interesting going on. Compare this with a patterned floor, table, etc., where our eyes can linger to determine how exactly the colors are changing. Emulating this behavior through under-sampling allows for efficiency improvements across the scene, with minimal qualitative impact when paired properly with super-sampling.

4 Project Methodology

4.1 Multiple Resolution Approach

In this project we utilize a multi-resolution approach to determine when to shoot more rays (*Lawlor and Genetti*, 2014). The basic premise involves first rendering the scene at a low resolution, some power of two fractionally lower than the target resolution. This lower resolution represents the under-sampled state of the scene render, where one ray represents multiple pixels of the render at the target resolution.

Information about the scene is collected from the render at a given resolution, ranging from information about where and how the ray hit the closest object, to the color of the resulting rendered image. This data is stored into a batch of individual textures matching the respective resolution. These texture files are then upscaled to match the next higher scene resolution.

At this new resolution tier, the renderer samples the textures of the previous resolution in order to determine whether to shoot new rays at the current resolution. This is accomplished by comparing the values of the textures respective to the current pixel with the values of the neighboring pixels. If there is a significant difference between the current pixel and the neighboring pixels, then it is assumed that something of interest is occurring within the local pixel block, and new rays should be shot at the current resolution. Conversely, if there is little difference between the current pixel and its neighbors, then there is no need to shoot new rays, and the results from the lower resolution render can be taken for the current resolution. This trace versus sample structure is what composes the core of the multi-resolution approach.

This process continues until the target resolution has been met, at which point all under-sampling has been accomplished. Super-sampling follows an equivalent process, simply by extending the approach to resolutions higher than the target resolution. The resulting renders can then be downscaled to the target screen resolution to reduce the affects of aliasing.

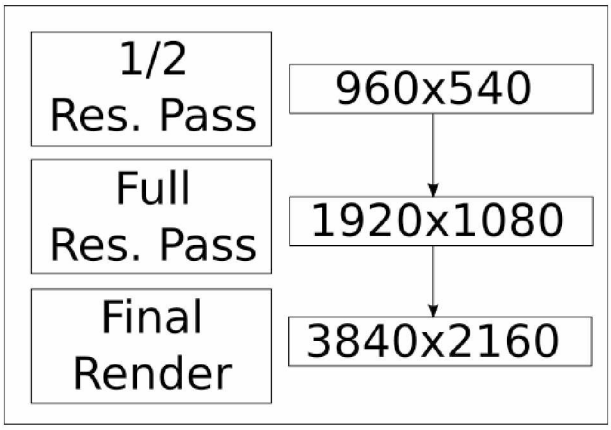


Figure 4.1: Summary of Multi-Resolution Approach

A summarized view of this process can be seen in Figure 4.1. The resolutions depicted in this figure are the ones that are utilized for the final project renders. We determined that three passes struck a balance of GPU efficiency improvement with little CPU overhead for managing the rendering pipeline.

4.2 Texture Sampling

Textures from previous resolutions can be used to determine whether new rays are necessary by sampling a given pixel location and comparing it with its neighbors for each texture. This raises two questions however: how are neighbors being defined, and in what manner are these pixel values being compared.

4.2.1 Choosing a Neighborhood

In general, when talking about a pixel neighborhood, we refer to a range of pixels around a given target center pixel that the program is rendering to. A neighborhood of four pixels describes the neighbors that are directly above/below and to the sides of the target pixel. A neighborhood of eight describes all pixels that are surrounding the center, and so on as seen in Figure 4.2.

The size a given neighborhood should be depends on how extensive of a range changes can occur while still influencing the target pixel sample. (*Lawlor and Genetti*, 2014) found that

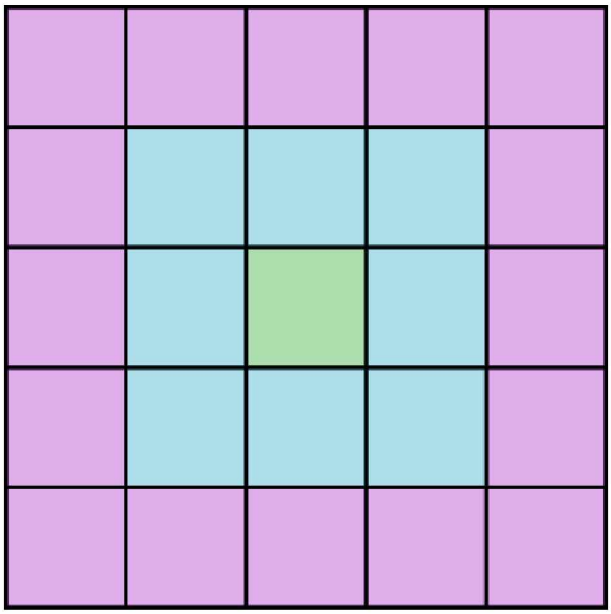


Figure 4.2: Basic Pixel Neigborhoods

increasing the size of the pixel neighborhood doesn't necessarily correlate with increased accuracy. Rather, it can introduce false positives that require detailed examination even though sources of change are far away from the target pixel. In the case of this project, a neighborhood composed of the eight pixels surrounding the target pixel was chosen for comparisons.

4.2.2 Normalization

Even with the proper neighborhood size being established, management of the differences with the target pixel must be defined. The goal is to have a normalized value that is the result of comparison between the target pixel and all of its neighboring pixels.

To start, a comparison between the target pixel and its neighbors is described as the difference between the expected value for the pixel from interpolation between opposing neighbors, and the actual information within the pixel. This process, seen in Figure 4.3, was found to be more effective than comparing the target pixel data against each neighbor individually.

Then comes the matter of normalizing the differences between each of the comparisons

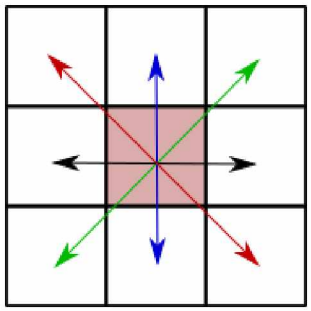


Figure 4.3: Pixel Comparison Process

into a single usable value. Two common methods are L1 and L2 normalization. L1 normalization simply takes the sum of absolute differences, whereas L2 normalization takes the sum of the squares of each difference, before taking the root of that value.

Through testing, we determined that taking the average of the L1 normalization of the differences produced the most reasonable metric for whether or not new ray samples were needed. So for each pair of neighboring pixels, the difference between their interpolated value and the target pixel was added to a sum, which was then averaged by the number of neighbors.

This difference can then be compared with an arbitrary threshold. If the normalized difference is larger than the threshold, then something of interest is occurring in the pixel neighborhood, and further ray samples are needed at the current resolution. Otherwise, data from the lower resolution textures can be taken and used for the current resolution. The resulting algorithm can be seen in Algorithm 1.

4.2.3 Heuristic Types

Finally, it is necessary to establish by what heuristic the target pixels and its neighbors are being compared. Image-space heuristics such as color returned by rays are fast and easy to work with, but previous research indicated flaws with its exclusive use, such as missing the edges of a white object against a white background. As such, this project also uses object-space heuristics such as where rays hit a given object in order to determine whether further rays are necessary. Information about each of these heuristics are stored as textures

Algorithm 1 Determine if Adaptive Sampling is Necessary

$diff_{L1} \leftarrow 0$
$diff_{L1} \mathrel{+}= abs(actualPixelData - lerp(leftNeighbor, rightNeighbor, 0.5)$
$diff_{L1} \mathrel{+}= abs(actualPixelData - lerp(topNeighbor, bottomNeighbor, 0.5)$
$diff_{L1} \mathrel{+}= abs(actualPixelData - lerp(topLeftNeighbor, bottomRightNeighbor, 0.5)$
$diff_{L1} \mathrel{+}= abs(actualPixelData - lerp(bottomLeftNeighbor, topRightNeighbor, 0.5)$
$avgDiff \leftarrow diff_{L1}/numNeighbors$
if $avgDiff.color > colorThreshold$ **then**
  return true
end if
if $avgDiff.distance > posThreshold$ **then**
  return true
end if
return false

for a given resolution, providing efficient access for future passes.

Specifically, when analyzing the render from a lower resolution, the algorithm will take the normalized input from the red, green, and blue color channels separately so that a steep gradient along any channel will indicate that super-sampling is necessary. The position data throughout a neighborhood is also collected and analyzed at this point, which will separately trigger the super-sampling process as necessary if the color heuristic did not.

This approach can be seen in Figure 4.4, where pixels highlighted in green represent the necessity of adaptive sampling based upon a color heuristic. Pixels highlighted in blue represent the necessity of adaptive sampling based upon a position heuristic. Due to the structure of the algorithm, where color is checked first, the presence of both colors in the image indicates that color alone is not a strong enough heuristic to capture all changes.

4.3 Tri-Adaptive Sampling

The core premise of this project is to modify the way in which rays are shot at the screen such that fewer texture samples are necessary to achieve the same degree of information. With the standard multi-resolution approach, one ray is shot at the center of each pixel. By offsetting the direction that rays are shot by half a pixel for every other row, a triangular grid is effectively established, as seen in Figure 4.5a. This data is still being stored in a 2D

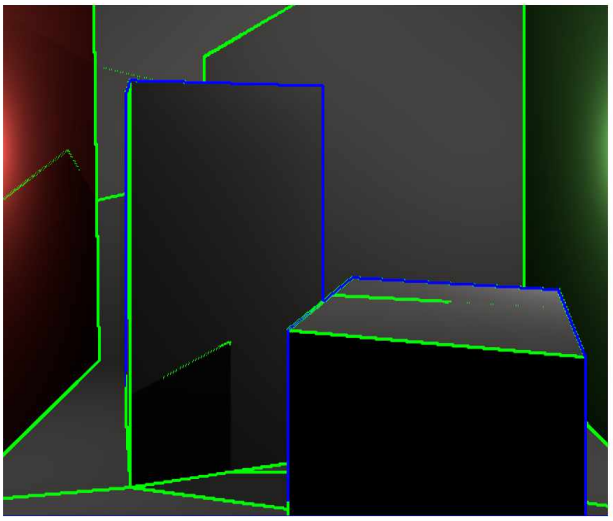
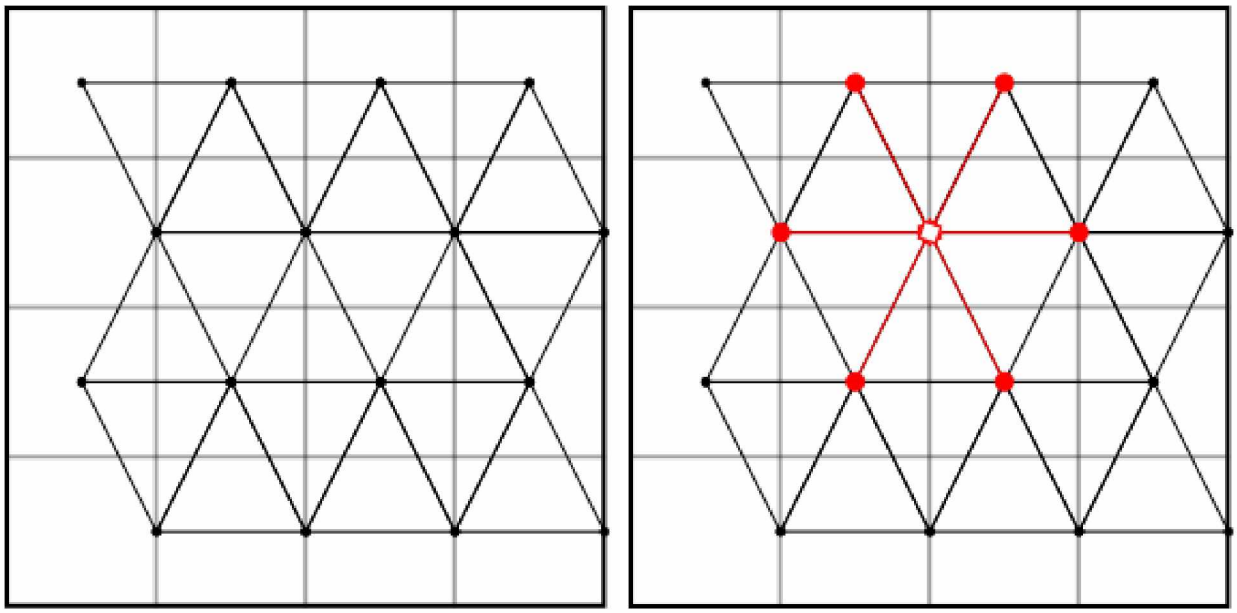


Figure 4.4: Heuristic Comparison

texture object, so conceptually it written to and read from a square grid as usual.

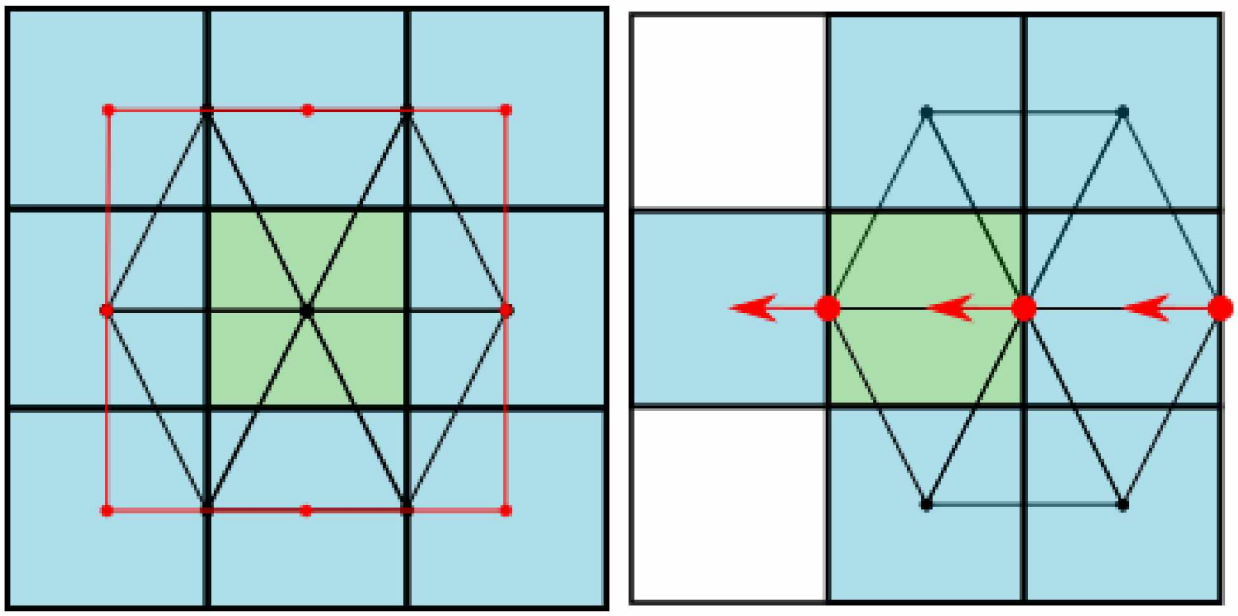
The goal of this process is to vary the distribution of pixel neighbors. As seen in Figure 4.5b, when analyzing the target pixel location, there are six neighbors that be compared against. With a distribution area between the four and eight neighbors of a traditional square sampler, we hypothesize that a similar degree of quality can be achieved in comparison with the eight pixel neighborhood. If this is the case, then a minor performance gain should arise from eliminating two samples from the process.

Now, the direct flaw in this system is that by offsetting the ray directions for every other row, creating a triangular grid, there would be obvious distortion in the final render that is displayed to the screen, which reads from a square grid. The solution to this is straightforward however, in that the program must simply map the triangular grid back to the square grid as seen in Figure 4.6. This mapping can be reasonably accomplished by interpolating the proper pixel color from the surrounding samples. So for each pixel in the offset row, simply interpolate between the two samples on either side, or include further samples from above and below the pixel to get an accurate color. This does, however, raise a question of whether these interpolations at the final stage will cause any type of performance penalty in comparison with the saved texture samples over several passes.



(a) Depiction of Triangular Overlay (b) Target "pixel" and six neighbors

Figure 4.5: Triangular Grid Overlay



(a) Neighborhood Area Comparison (b) Sampling Neighborhood From The 2D Grid

Figure 4.6: Triangular Neighborhood Mapping

5 Data Metrics and Results

5.1 Project Setup

5.1.1 Software

The ray tracing rendering engine used for this project was built using Microsoft's DirectX Ray Tracing (DXR) API (*Hargreaves and Rhyn*, 2019). This API is designed to run through existing DirectX 12 engines, introducing various acceleration structures alongside ray tracing functionality that is intertwined with HLSL shader structures (*Stich*, 2019). Notably, it is this DXR API that allows developers to take advantage of the hardware-level ray tracing on NVIDIA RTX compatible GPU cards. As such, usage of this API allows for optimal performance for the final data metrics, and a closer resemblance to how modern real-time applications will run.

In addition, this project makes use of the Falcor framework, which is an open-source framework by NVIDIA that is intended for rapid development prototyping (*NVIDIA*, 2018b). This framework creates an abstraction layer atop the DirectX 12 and Vulkan engines, encapsulating the functionality into neat pipeline arrangements that are easy to modify and manage. This framework also allows for simplified scene construction through the use of traditional object file formats in a specific scene description file. This allows for data metrics to be collected across multiple scenes.

5.1.2 Hardware

Once the development of the renderer was finished, we ran the program on a Asus ROG laptop with a Intel Core i7-8750H CPU and a GeForce RTX 2070 GPU card. As such, recorded performance times represent the capabilities on a machine with hardware-level ray tracing.

5.2 Scene Setup

The scenes chosen for data collection are as follows. The Cornell Box is a staple of rendering research, and as such is useful for testing a simple scene without a great deal of geometry. Point lights were added to the scene to help showcase the algorithm through shadows in the resulting render. The Pink Room scene is included with the Falcor code distributions, and provides a set of more complex scene geometry to be used for testing.

5.3 Performance

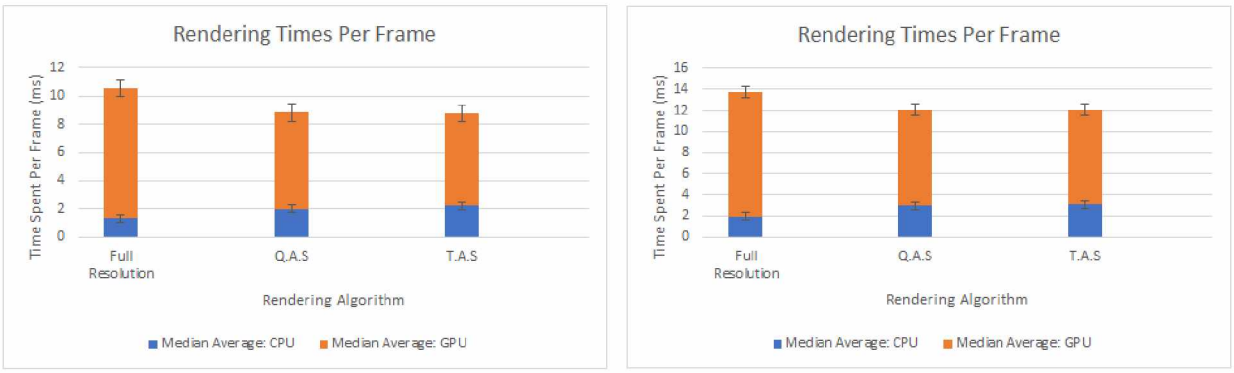
Performance as a whole is being measured through frames-per-second (FPS). We utilize Falcor and DirectX's profiling capabilities to pull out data regarding FPS as well as the time it takes for individual resolution passes to process. We use the latter data to determine where the dominate computation time lies. These metrics allow us to compare the Tri-Adaptive sampling method with traditional Quad-Adaptive sampling, as well as against a render of the scene at full resolution without any adaptive-sampling.

For a given scene, performance data is collected for 1024 frames, after which the data is exported for analysis in what is being called a sample. For each sample, the average FPS is pulled out, as well as the average time per frame spent on rendering between the CPU and GPU. Five samples are collected for each of the rendering pipelines: Full Resolution, Quad-Adaptive Sampling, and Tri-Adaptive Sampling.

Table 5.1: Median Values for Sample FPS Averages

	Full Resolution	Quad-Adaptive Sampling	Tri-Adaptive Sampling
Cornell Box	106.47	144.10	150.01
Pink Room	84.33	108.11	109.23

Table 5.1 shows the median values for the five samples of each pipeline for all scenes tested. As expected, the adaptive sampling algorithms outperform the full resolution renderer by a sizeable margin. Specifically, the Tri-Adapative renderer provides a performance boost of 40.9% over the naive approach for the Cornell Box scene, and a performance boost of 29.5%



(a) Computation Time For Cornell Box Scene (b) Computation Time For Pink Room Scene

Figure 5.1: Scene Render Timing Breakdown

for the Pink Room scene. This is largely attributed to the use of an adaptive sampling algorithm in general as the traditional Quad-Adaptive renderer provides performance boosts of 35.4% and 28.2% respectively over the naive approach.

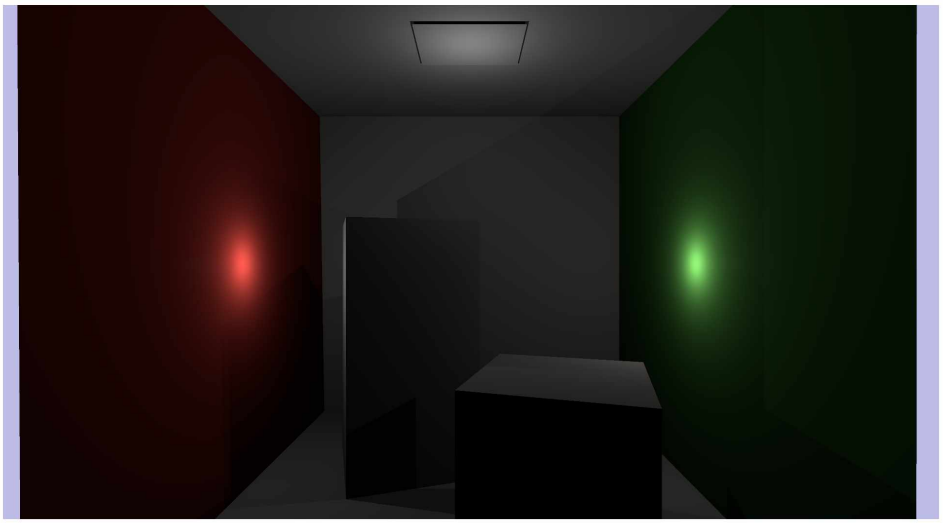
Analysis between the two adaptive sampling algorithms indicates a statistically significant difference per a one-tailed t-test with a significance value of 5%. With this, the null hypothesis that there is no performance difference between the algorithms can be rejected. Comparing the median FPS values, there is a 4.1% improvement in efficiency using the Tri-Adaptive algorithm while rendering the Cornell Box scene, and a 1.04% improvement in efficiency while rendering the Pink Room scene.

Examining these results more closely, there is an interesting distinction between the rendering algorithms in their timing breakdowns. Figure 5.1 shows the average amount of time spent by the CPU and GPU in order to render a single frame. As before, the values used are the medians of the five sample averages for each algorithm. These figures show that, in general, the adaptive sampling algorithms have more CPU overhead due to additional rendering passes and the associated overhead of manipulating textures. This CPU overhead is seemingly slightly higher for the Tri-Adaptive Sampling algorithm in comparison with the Quad-Adaptive Sampler, however it requires less time per frame for the GPU side.

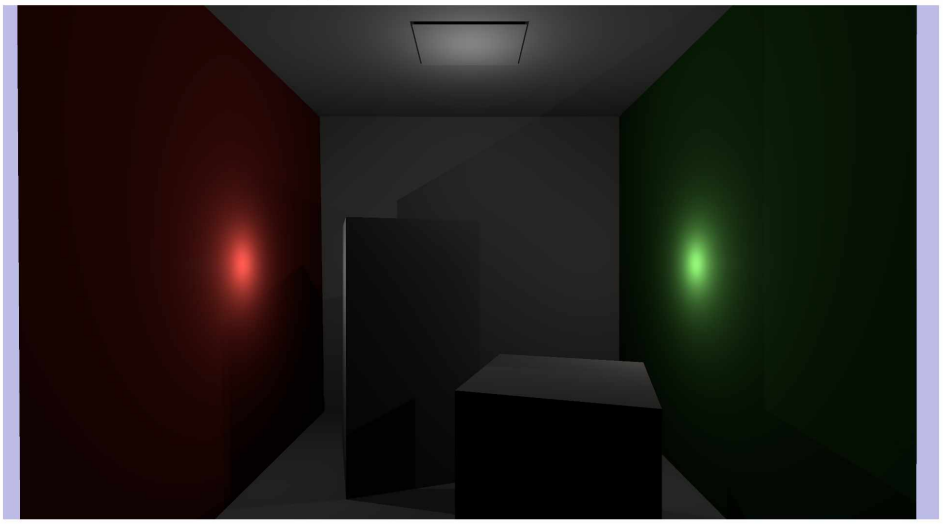
5.4 Quality

We determine the quality of the algorithm output by comparing the resulting images of the adaptive sampling renders with a ground truth image. This ground truth is considered to be the absolute state of the image, and any deviations from it indicate deficiencies in quality as a result of the algorithms. We define the ground truth in this instance as the result of rendering the scene at maximum quality without any adaptive sampling algorithms. This image is then compared against the Quad-Adaptive Sampler and the proposed Tri-Adaptive Sampler to determine whether either adaptive algorithm results in loss of image quality.

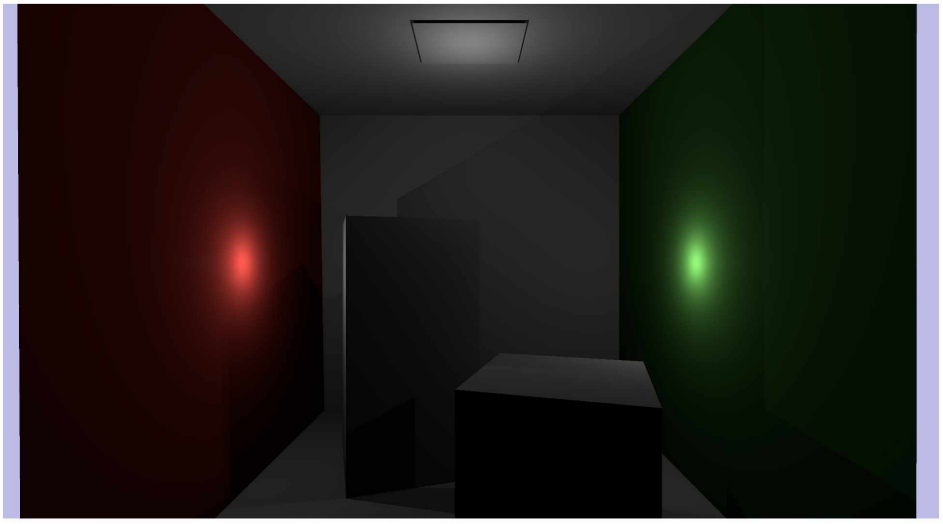
The resulting scene renders can be seen in Figure 5.2 and Figure 5.3. Offhand, there appears to be very little loss in image quality, if any at all. This is supported by Figure 5.4, which are images generated by taking the absolute difference between two input images in a photo editing program, and then adjusting the brightness curve such that differences are clear. The Cornell Box diff image had its brightness multiplied by roughly 23x, and the Pink Room diff image was multiplied by roughly 11x. Examining brightness histograms in Figure 5.5 for both scenes gives a clear indication of the similarity of the images, with 93.3% of pixels having no appreciable brightness in the Cornell Box scene, and 89.3% in the Pink Room scene. With any pixel coloring and brightness only occurring when there are differences between the images, the generated diff images indicate extremely little difference between the full resolution render and the Tri-Adaptive render, meaning that there is minimal loss in quality. Similar results for the Quad-Adaptive render indicate that there is little difference between the algorithms in terms of appearance as both maintain similar quality to the ground truth image.



(a) Full Resolution Render



(b) Quad-Adaptive Sampling Render



(c) Tri-Adaptive Sampling Render

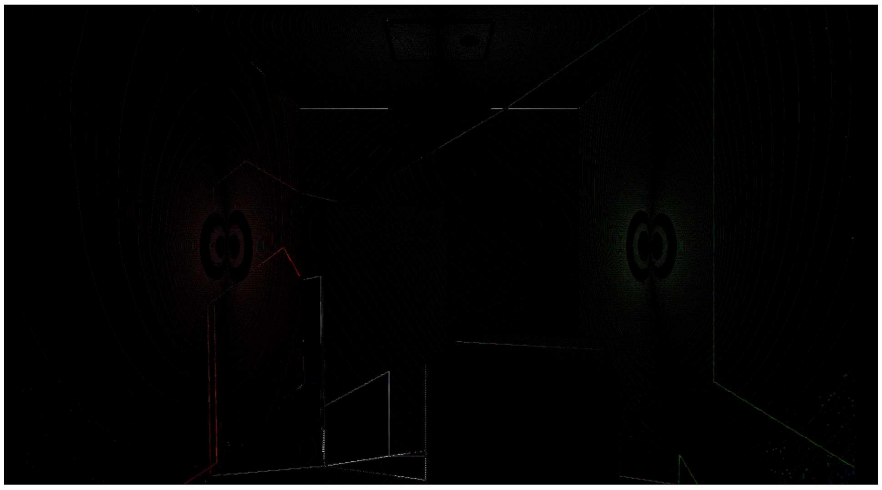
Figure 5.2: Cornell Box Renders

(a) Full Resolution Render

(b) Quad-Adaptive Sampling Render

(c) Tri-Adaptive Sampling Render

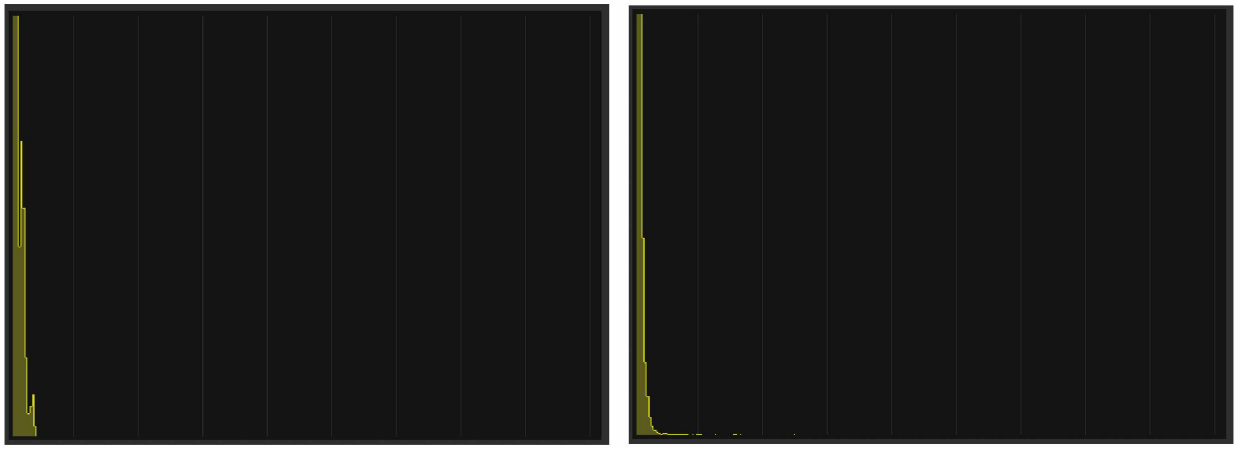
Figure 5.3: Pink Room Renders



(a) Cornell Box Diff Image (Brightness x23)

(b) Pink Room Diff Image (Brightness x11)

Figure 5.4: Diff Image Results Between Full Resolution and Tri-Adaptive Renderers



(a) Cornell Box Diff Image Brightness Histogram (b) Pink Room Diff Image Brightness Histogram

Figure 5.5: Histogram Showing Diff Image Brightness Values

6 Conclusion

Adaptive sampling is a process that is simultaneously capable of improving render quality through natural anti-aliasing, while reducing the cost necessary to render a scene by shooting few rays in areas of little change. By rendering the scene at various resolutions ranging from coarse to fine, it is possible to manually control how finely sampled the final scene render is. Coarser resolution grids are capable of eliminating unnecessary rays while the finer resolution grids are capable of anti-aliasing and preventing loss in scene quality.

We proposed a modification to the traditional multi-resolution adaptive sampling process, where a triangular grid is utilized for shooting rays and sampling textures. This changes the distribution of texture samples, allowing for fewer samples when determining whether to shoot new rays, while maintaining a similar degree of quality to the traditional approach.

Results on an RTX GPU indicate that our approach improves performance by 29-40% over the naive render, and by 1-4% over the quad-adaptive renderer, all while maintaining a similar degree of quality to the ground truth image. We believe that the performance difference is dependent upon the complexity of the scene, as more detailed geometry and textures will cause both adaptive sampling algorithms to trigger regardless, reducing the performance difference between the two. In addition, a decision was made to eliminate indirect lighting from the program to reduce noise in resulting renders. Inclusion of more complex ray tracing techniques such as indirect lighting may increase the performance difference as the time saved per frame would hypothetically increase alongside the performance cost of a new ray. Future research should aim to examine the impact of complex geometry and complex textures on performance, as well as the effect of more complex ray tracing algorithms including components such as indirect lighting and volumetric scattering.

References

Binks, D. (2011), Dynamic resolution rendering article.

Bongjun Jin, B. C. C. P. W. L. S. J., Insung Ihm (2009), Selective and adaptive supersampling for real-time ray tracing, doi:10.1145/1572769.1572788.

Cook, R. L. (1986), Stochastic sampling in computer graphics, *ACM Trans. Graph.*, *5*(1), 51–72, doi:10.1145/7529.8927.

Cook, R. L., T. Porter, and L. Carpenter (1984), Distributed ray tracing, *SIGGRAPH Comput. Graph.*, *18*(3), 137–145, doi:10.1145/964965.808590.

Genetti, J., and D. Gordon (1993), *Ray Tracing With Adaptive Supersampling in Object Space.*

Hachisuka, T., W. Jarosz, R. P. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. W. Jensen (2008), Multidimensional adaptive sampling and reconstruction for ray tracing, *ACM Trans. Graph.*, *27*(3), 1–10, doi:10.1145/1360612.1360632.

Hargreaves, S., and J. v. Rhyn (2019), Announcing microsoft directx raytracing!

He, Y., Y. Gu, and K. Fatahalian (2014), Extending the graphics pipeline with adaptive, multi-rate shading, *ACM Trans. Graph.*, *33*(4), doi:10.1145/2601097.2601105.

Kim, Y., W. Seo, Y. Kim, Y. Lim, J.-H. Nah, and I. Ihm (2016), Adaptive undersampling for efficient mobile ray tracing, *The Visual Computer*, *32*(6), 801–811, doi:10.1007/s00371-016-1251-y.

Křivánek, J., I. Georgiev, T. Hachisuka, P. Vévoda, M. Šik, D. Nowrouzezahrai, and W. Jarosz (2014), Unifying points, beams, and paths in volumetric light transport simulation, *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, *33*(4), doi:10/f6cz72.

Lake, A., D. Bois, and L. Reznikov (2019a), Get started with variable rate shading on intel® processor graphics.

Lake, A., F. Strugar, K. Gawne, and T. McFerron (2019b), Use variable rate shading (vrs) to improve the user experience in real-time game engines.

Lawlor, O. S., and J. D. Genetti (2014), Exploiting spatial redundancy with adaptive pyramidal rendering.

McFerron, T., and A. Lake (2018), Dynamic resolution rendering update for microsoft directx* 12.

Mitchell, D. P. (1991), Spectrally optimal sampling for distribution ray tracing, *SIGGRAPH Comput. Graph.*, *25*(4), 157–164, doi:10.1145/127719.122736.

NVIDIA (2018a), Nvidia turing gpu architecture, *Tech. rep.*

NVIDIA (2018b), Falcor.

Stich, M. (2019), Introduction to nvidia rtx and directx ray tracing.

van Rhyn, J. (2019), Variable rate shading: a scalpel in a world of sledgehammers.

Whitted, T. (1980), An improved illumination model for shaded display, *Commun. ACM*, *23*(6), 343–349, doi:10.1145/358876.358882.

Won-Jong Lee, Y. S. S. R. I. I., Seok Joong Hwang (2016), Adaptive multi-rate ray sampling on mobile ray tracing gpu, doi:10.1145/2999508.2999523.

Xiao, K., G. Liktor, and K. Vaidyanathan (2018), Coarse pixel shading with temporal supersampling, in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '18, Association for Computing Machinery, New York, NY, USA, doi:10.1145/3190834.3190850.